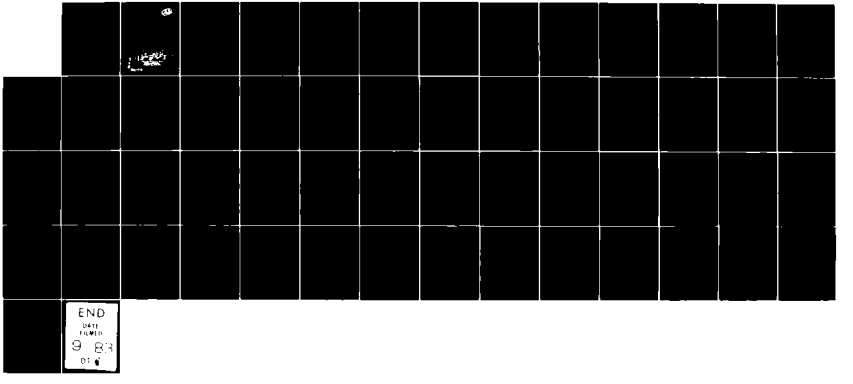
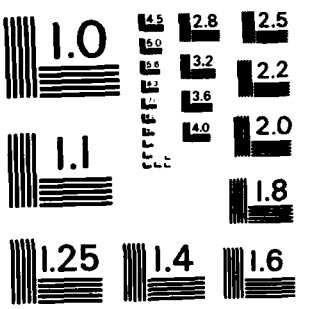
AD-A132 524 KNOWLEDGE REPRESENTATION AND RETRIEVAL FOR NATURAL LANGUAGE PROCESSING(U) ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE A M FRISCH ET AL. DEC 82 TR-104 1/1

UNCLASSIFIED N00014-82-K-0193 F/G 6/4 NL



END
DATE FILMED
9 83
DTIC



Knowledge Representation and Retrieval for Natural Language Processing

Alan M. Frisch
James F. Allen

Computer Science Department
The University of Rochester
Rochester, NY 14627

TR 104
December, 1982

Knowledge Representation and Retrieval for Natural Language Processing

Alan M. Frisch
James F. Allen

Computer Science Department
The University of Rochester
Rochester, NY 14627

TR 104
December, 1982

Abstract

We are building a computer system called ARGOT that acts as a computer operator conversing with a computer user. Since performance in this domain requires ARGOT to have efficient access to a large body of diverse knowledge, a major part of our research effort has been focussed on issues of knowledge representation and retrieval. This paper describes ARGOT's representation language, the retriever used to access a knowledge base of sentences of the language, and how their design has been influenced by the task domain and system organization.

Preface

Chapter 2 of this report is a slightly updated version of a paper presented at the 6th Conference on Automated Deduction (Frisch and Allen, 1982). Chapter 3 is an updated extension of a paper presented at the 20th Annual Meeting of the Association for Computational Linguistics (Allen and Frisch, 1982). Though the two papers have appeared separately and can be read in isolation, we feel that the significance of each paper is enhanced when considered in the context of the other. In Chapter 1, the introduction, we set the stage for the two papers by explaining how they fit into the greater context of our overall research efforts.

Acknowledgements

The work reported here has taken place over an extended period of time within the setting of the natural language group in the University of Rochester Computer Science Department. We have benefited greatly from their support and assistance. This group has included Mark Giuliano, Pat Hayes, Dave Lewy, Diane Litman, Steve Small, Lokendra Shastri, and Marc Vilain. Andy Haas and Don Perlis have been sources of intellectual stimulation, providing assistance and insistence during all phases of our research.

The presentation of our work has been improved as a result of suggestions made by Pat Hayes, Diane Litman, Dan Russell and the department's Artificial Intelligence Study Group.

The implementation of ARGOT's knowledge base module has profited from the efforts of several people. Mark Guiliano's contribution to the implementation of the HORNE logic programming system facilitated many features of the knowledge base. Dave Lewy, Diane Litman and Marc Vilain have demonstrated unreasonable patience in using the various incarnations of the knowledge base. Without them, the system would still be bug-laden and unfriendly.

This research has been supported by NSF Grant IST-80-12418, DARPA Grant N00014-82-K-0193 and ONR Grant N00014-80-C-0197.

Contents

Chapter 1

Introduction

This paper attacks some of the problems of knowledge representation and retrieval that we have encountered in designing and constructing a system that can partake in an extended English dialogue. The system, called ARGOT, consists of a number of interacting modules, one of which is called the knowledge base (KB). The KB stores sentences of the representation language and provides facilities for the other modules to retrieve this knowledge. This paper is concerned with retrieval, its relationship to the representation language on which it operates, and the representation and retrieval capabilities that are necessitated by ARGOT's task domain and system organization.

Though the past few years have seen a growing concern for formalization in the study of knowledge representation, little has been done on the formalization of the retrieval processes that operate on these representations. Chapter 2 proposes to remedy the situation by viewing retrieval as a limited inference process operating on a body of stored knowledge. A method for using first-order predicate calculus (FOPC) to specify a limited inference mechanism of this sort is put forth. This technique is then used to specify a retriever that, for example, performs typical semantic-network inferences such as inheritance but not arbitrary logical inferences such as modus ponens.

Using this viewpoint of knowledge retrieval as limited inference, Chapter 3 examines semantic-network representations and their retrievers. A semantic network is viewed as a representation that uses specialized notation to encode certain epistemological information, such as type hierarchies. A semantic-network retriever has specialized techniques for dealing with the specialized notation. In a sense, the retriever *knows* something about the special notation. But exactly what does it know? We set out to answer this question by laying out a set of axioms that encode this knowledge.

Chapter 2 and Chapter 3, though intricately related, are each self-contained. The remainder of this introductory chapter serves to explicate this relationship and put the two chapters in the greater context of our overall research efforts.

Section 1.1 briefly overviews the ARGOT system. Its thrust is that since the knowledge base in ARGOT plays a much different role than a knowledge base in a typical question-answering system, a more expressive representation language and a different set of retrieval facilities are necessary. Section 1.2 presents a glimpse at the knowledge representation language used in ARGOT. Section 1.3 concludes this chapter with an intuitive sketch of the retrieval facilities of the knowledge base module.

1.1 ARGOT and its Knowledge Base

ARGOT (Allen, Frisch and Litman, 1982), plays the role of a computer operator partaking in a dialogue with a computer user. The following dialogue, a slightly cleaned up version of an actual dialogue, has served to motivate our work:

(1) User: Could you mount a magtape for me?
(2) It's T376.
(3) No ring please.
(4) Can you do it in five minutes?
(5) Operator: We are not allowed to mount that magtape.
(6) You will have to talk to the head operator about it.
(7) User: How about tape T241?

The dialogue illustrates that ARGOT is not a simple question-answering system. First of all the user has goals beyond that of getting some information. In this case the user has the goal of reading a magtape and his plan for doing so contains the subplan of requesting the operator to mount a tape. In addition, the operator can perform non-linguistic actions - in this case he is able to mount a magtape.

The dialogue does not consist of a successsion of independent question-answer pairs. A speaker can make several utterances in a row, each of which may only be meaningful in the context of other utterances. Uterances may be elliptical, ungrammatical, and non-literal. For example the first utterance is a request for the operator to mount a magtape, not a request for the operator to inform the user of his ability to mount a magtape.

For our present purpose there is not much need to investigate the dialogue participation task in greater detail. Cohen, Perrault and Allen (1982) analyze

the task and approaches to it, while Allen, Frisch and Litman (1982) provide details about how ARGOT handles the problems.

ARGOT is divided into many concurrent communicating modules in addition to the KB. The current implementation contains three major modules: a task goal reasoner, a communicative goal reasoner, and a linguistic reasoner. Each of these modules is intended to perform both recognition and generation, but our research to date has focussed primarily on recognition.

As in many AI systems, ARGOT's knowledge base (KB) stores the explicitly represented knowledge used by the system and provides facilities for accesssing the knowledge. However, the KB plays a different role in ARGOT than in a typical question-answering system. A typical question-answering system proceeds by translating the user's natural language query into a query in the system's knowledge representation language. The KB contains information about the domain of the question, on the basis of which it attempts to infer a response. In the extreme case, the KB is used solely for inferring an answer to the query; the linguistic processing that translates the natural language query is totally independent of the KB.

In contrast, all modules in ARGOT make extensive use of a common knowledge base. As a consequence of this approach, a great deal of generality is required of the KB; the representation language and the retrieval facilities must be useful for a range of processing that includes both parsing and response planning.

The sentences stored in the KB not only represent the knowledge associated with magtapes and the doings of a computer room, but also the knowledge necessary to deal with language. In addition to knowledge of linguistic structure, this includes knowledge of physical, mental and linguistic actions. how these actions are involved in plans, what computer users do, and what they expect of computer operators.

Unlike the KB of a some question-answering systems, the ARGOT KB cannot be an unrestricted inference engine. It is not be feasible to control a powerful inference engine without specializing it to a specific domain. Rather, the powerful domain-dependent inference must be done by the modules that have been tailored for the various types of analysis. The KB provides the facilities

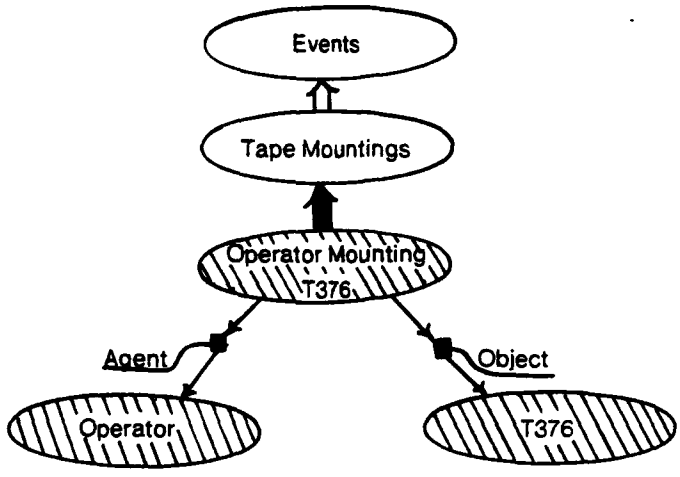
for these modules to retrieve the stored knowledge that they use to do their analyses. The KB retriever is limited to performing those inferences that it can control adequately. These include the type of inferences that semantic networks traditionally have been built to support. A module can compensate for a KB that is efficient but inferencially weak by making inferences of its own, but it is hard for a module to compensate for a KB whose excessive inferencial power leads to inefficiency.

1.2 The Representation of Knowledge in ARGOT

Following Brachman (1979), our representation is constructed of two levels: the epistemological level and the conceptual level. The epistemological level provides a set of knowledge structuring primitives that are used to construct all conceptual level entities (e.g., action, time, and belief). Each level of the representation provides primitive symbols (particular predicate symbols, function symbols, and constant symbols) which can then be combined using the notation of FOPC. By inheriting the logical connectives and quantificational structure of FOPC, the resulting representation language is rather expressive.

1.2.1 The Epistemological Level of Representation

The episemological level of the representation supplies a fixed set of predicates which are the knowledge-structuring primitives out of which all representations are built. The choice of knowledge-structuring primitives has been motivated by the study of semantic networks. For instance, where a semantic network such as Brachman's (1979) might have



we would have

SUBTYPE(Tape-Mountings,Events)
TYPE(Operator-Mounting-T376,Tape-Mountings)
ROLE(Operator-Mounting-T376,Agent,Operator)
ROLE(Operator-Mounting-T376,Object,T376)

Notice that the SUBTYPE predicate corresponds to the unshaded double arrow, the TYPE predicate to the shaded double arrow, and the ROLE predicate to the single arrow. Our constants are sorted into individuals (e.g.: Operator-Mounting-T376, Operator, T376), types (e.g.: Events, Tape-Mountings) and rolenames (e.g.: Agent, Object). These sorts correspond somewhat to the shaded oval, unshaded oval, and shaded box of the network.

Chapter 2 fully describes and axiomatizes the epistemological level of the representation and discusses it's relationship to semantic networks.

1.2.2 The Conceptual Level of Representation

With the exception a brief discussion of time in Section 3.7.1, this section is the paper's sole encounter with the conceptual level of representation. It's purpose is to give the flavor of some diverse studies in this area and show how they are related to each other and to our use of language.

The representation of actions is crucial to a dialogue participant for two reasons. The first is that the participant must be able to represent the meaning

of utterances that refer to actions (e.g., "Can you mount a magtape for me?" refers to the action of mounting). The second is that it is advantageous to model the language comprehension and production processes as purposeful. planned action (e.g., uttering "Can you mount a magtape for me?" is a requesting action). However, existing models of action, most notably the state-space approach (e.g. (Fikes and Nilsson, 1971)), appear inadequate for the above purposes. Since a major deficiency with the existing models is an inadequate treatment of time, we first turn our attention to this issue.

An interval-based temporal logic and its associated inference processes have been defined (Allen, 1981a). Rather than using a global time line, the representation employs a hierarchical set of reference frames. A particular interval is known by its location relative to the reference frames and other intervals. This is particularly important in a dialogue system for most temporal knowledge does not have a precise time. The hierarchical reference frames facilitate efficient updating of the representation of the current moment (i.e., now). Allowing time intervals to extend indefinitely into the past or future supports limited default reasoning about the persistence of properties.

This representation of time has been used to produce a general model of events and actions (Allen, 1981b). Rather than concentrating on how actions are performed, as is done in the problem-solving literature, this work examines the set of conditions under which an action or event can be said to have occurred. In other words, if one is told that a certain action occurred, what can be inferred about the state of the world?

The approach to this problem is to view an occurence of an event as a partial description of the world over some time interval. Actions are then defined as that subclass of events that are caused by agents. This is in contrast to the state-space view of an action as a function from one world state to a succeeding world state. Our approach enables the representation of actions that describe inactivity (e.g., standing still), preserving a state (e.g. preventing your television from being stolen), and simultaneous performance of simpler actions (e.g., talking while juggling). A crucial aspect of our view of an action is that it does not necessarily involve any specific physical activity.

Representing actions, particularly speech acts, requires the representation of beliefs. For example, consider the speech act of the user requesting at time t

that the operator mount tape T376. The effect of the action is that, during the interval starting after t and extending indefinitely into the future, the operator believes that the user intends for him to mount tape T376.

A model of belief has been developed that treats BELIEVE as a predicate on an agent and a sentence of the representation language. To do this, there must be a name for every sentence in the language. Perlis (1981) and Haas (1982) have introduced naming schemes that provide enough expressiveness to deal with traditional representational requirements such as quantifying in. (For purposes of this paper, we will name a sentence α by 'α').

Difficulties arise when we try to relate the name of a formula to the formula it names. For example, consider representing "John believes something that is false" as $\exists x$ BELIEVE(John,x) $\wedge$ ~TRUE(x). We would like the predicate TRUE to be such that for any sentence α, TRUE('α') is true exactly when α is true; that is, TRUE('α') $\leftrightarrow$ α. Unfortunately, such an axiom schema is inconsistent. However, Perlis (1981) has defined an axiom schema that usually behaves like the above schema but is provably consistent.

Haas (1982) has used this formulation of belief to predict an agent's action by constructing plans that can include mental actions. His treatment of belief and action does not suffer from the problem of the possible worlds approach – e.g., Moore (1979) – that an agent believes all consequences of his beliefs.

1.3 Knowledge Retrieval

The Knowledge Base (KB) provides a set of retrieval facilities that is the sole access that the system has to the sentences stored in the KB. Since retrieval must respect the semantics of the representation, it is viewed as inference. However, this inference must be limited because retrieval must terminate, and must do so in a reasonable amount of time. Chapter 2 shows how a limited inference engine suitable for knowledge retrieval can be given a formal, non-procedural specification in a meta-language and how such a specification can be efficiently implemented. The retriever developed in that chapter is a simplified version of the one employed by the ARGOT KB.

The capabilities and limitations of the retriever can be thought of intuitively as follows. A set of axioms dealing solely with the epistemological primitives is built into the retriever. For example, three of these axioms are:

$\forall t_1.t_2.t_3$ SUBTYPE(t_1.t_2) $\wedge$ SUBTYPE(t_2.t_3) $\rightarrow$ SUBTYPE(t_1.t_3)
(SUBTYPE is transitive.)

$\forall o.t_1.t_2$ TYPE(o.t_1) $\wedge$ SUBTYPE(t_1.t_2) $\rightarrow$ TYPE(o,t_2)
(Every member of a given type is a member of its supertypes.)

$\forall$ x,r,y,y' ROLE(x,r,y) & ROLE(x,r,y') $\rightarrow$ y = y'
(Role fillers are unique)

Through these built-in axioms, the retriever "knows about" all of the episemological primitives. The retriever's power comes from the fact that it can, for the most part, reason completely with the built-in axioms. Its limitations arise because it only partially reasons with the sentences stored in the KB. The retriever also has knowledge of how to control inferences with the built-in axioms. In this manner, the retriever only performs those inferences for which it has adequate control knowledge to perform efficiently.

The KB derives a considerable amount of its inferential power by storing its sentences is a normal form. Frisch (1981) has developed a normal form for languages with a belief operator, as well as a method for efficiently performing certain inferences on normalized sentences.

Chapter 2

Knowledge Retrieval as Limited Inference

Abstract

Artificial intelligence reasoning systems commonly employ a knowledge base module that stores a set of facts expressed in a representation language and provides faciities to retrieve these facts. A retriever could range from a simple pattern matcher to a complete logical inference system. In practice, most fall in between these extremes, providing some forms of inference but not others. Unfortunately, most of these retrievers are not precisely defined.

We view knowledge retrieval as a limited form of inference operating on the stored facts. This paper is concerned with our method of using first-order predicate calculus to formally specify a limited inference mechanism and to a lesser extent with the techniques for producing an efficient program that meets the specification. Our ideas are illustrated by developing a simplified version of a retriever used in the knowledge base of the Rochester Dialog System. The interesting property of this retriever is that it performs typical semantic network inferences such as inheritance but not arbitrary logical inferences such as modus ponens.

2.1 Introduction

Artificial intelligence reasoning systems commonly employ a knowledge base module (KB) that stores a set of facts expressed in a representation language and provides faciities to retrieve these facts. Such a module then is used by the reasoner in performing its task-specific inference. In designing a knowledge base, it is important to maintain a fine balance between efficiency and usefulness. The module should perform only those inferences for which it has adequate control knowledge to perform efficiently. For instance, we present a KB that performs inheritance as found in semantic networks but not general logical inference such as modus ponens.

Thus, the fundamental issue in designing a KB retriever is how to limit inference. One possibility is to limit the expressive power of the representation

language so that an efficient search space is produced. This is a bad idea as the reasoner may depend on being able to represent and retrieve arbitrary facts. The other alternative is to leave the representation language completely general and to limit the inferences that could occur during retrieval. This paper describes our methodology for specifying such a *limited inference* process.

The methodology uses first-order predicate calculus (FOPC) as the specification formalism. The use of FOPC notation in a representation language is not uncommon (Nilsson, 1980). And, as the study of logic programming (Kowalski, 1979) tells us, a retrieval algorithm can be specified in logic. Though the representation logic and the retrieval logic are different. they are not independent! In particualr, the representation logic must have a well-defined semantics in order to be useful to the reasoner. The retrieval logic must respect this semantics by specifying only valid inferences. An example of a relationship between the two logics that meets this criteria is expressed by the statement, "The sentence '$\alpha \wedge \beta$' can be retrieved if and only if the sentence 'α' and the sentence 'β' each can be retrieved."

We continue this paper by refining our notion of retrieval and then present our method of using FOPC to give a functional specification of a retriever. We illustrate these methodological points by a two-part presentation of a simplified version of the knowledge base module of the Rochester Dialog System (Allen, Frisch and Litman, 1982). The first part of the presentation considers a retrieval based on logical inference and the second part extends the system to handle semantic-network inference. We conclude with a brief discussion of techniques for implementing an efficient program that meets the specification.

This paper uses conventional notation for FOPC. The symbol used for logical implication or entailment is $\Rightarrow$ and the statement it forms is called a *sequent*. The symbol used for logical equivalence or bi-directional entailment is $\equiv$ and the statement it forms is called an *equivalence*. As is traditional, the order of precedence of the logical symbols from highest to lowest is; $\sim$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\forall$, $\exists$. In logical formulae, variables appear in lower case while constant symbols (predicates and functions) appear in upper case. Greek letters are used in schemata as variables that range over formulae or constant symbols.

2.2 What Is Retrieval?

At any point the KB will have a set of facts called its base facts, on the basis of which it responds to queries made by the reasoner. The issues addressed by this section are what form a query should take and what the retriever should do to respond to the query. We will characterize two extreme positions that can be taken and then present our position as lying somewhere between them.

At one extreme the set of base facts can be treated as an abstract data structure and queries as calls to utilities for examining it. This approach neither prevents a reasoner from performing any kind of operations it wants nor commits a reasoner to any kind of operations it doesn't want. Yet such a retriever doesn't provide much assistance. Presumably, there are operations that can be factored out of the reasoner to be done automatically and efficiently by the retriever on each query.

At the other extreme the set of base facts can be treated as a representation and a query as a request to determine if a given sentence logically follows. This approach has several difficulties that are caricatured by the fact that there is no assurance that the retrieval process will ever terminate. Allocating the retriever such power is putting the muscle of the system in the wrong place. It is the reasoner that is specialized for appropriately controlling inference in its task domain.

Our viewpoint is that retrieval must respect the semantics of the representation language and is therefore inference. A query is a request for the retiever to attempt to infer a specified sentence of the representation language. (For purposes of this paper we will assume that the sentence is closed. It is a trivial extension (Green, 1969; Luckham and Nilsson, 1971) to consider a query as an atempt to prove an instance, or even all instances, of an open sentence and to return the instance.) If this attempt to prove the querried sentence must be guaranteed to terminate then either the representation language must be severely restricted or the inference engine must be severely restricted and thus incomplete. Restricting the representation language would be a serious mistake. The role of a representation is to define the set of valid inferences that *could* be made, not those that *are* made. Even if the retriever only makes a small portion of all valid inferences the remaining possibilities must be available for the reasoner to consider. Our refusal to restrict the

representation language leaves us with the problem of designing a limited inference engine.

One common approach to limiting inference is to restrict the amount of resources used in the computation (Norman and Bobrow, 1975; Bobrow and Winograd, 1977; Robinson and Sibert, 1982). This can be done by restricting computation time, the total number of inference steps taken, or the depth of the inference. These approaches are unsuitable for knowledge retrieval because they limit all forms of inference uniformly. For example, if inference is limited to a depth of 5, then properties cannot be inherited down 6 levels of a type hierarchy. In general, there may be some kinds of inference that we want to be computed completely and others that we want to be ignored completely. A methodology for limiting inference should provide the knowledge base designer with enough control to pick and choose the inferences that he wants done.

Another class of limited inference systems are the incomplete theorem provers that are fairly common in the literature - for example, Brown's (1978) system. Typically, these systems are not guaranteed to terminate. They often have the undesirable property that the prover has a base fact yet cannot respond to the query that consists of that very fact. It is not surprising that these inference systems are ill-suited for knowledge retrieval; they were not designed for that purpose.

For a while we attempted to design a specialized resolution theorem prover suited for knowledge retrieval. We tried to conjure up a scheme for limiting the resolutions in the search space. The attempt failed, again because it was so difficult to limit inference and yet have all base facts retrievable.

It is instructive to consider why it is so difficult to restrict a traditional proof system without infringing on the retrievability of the base facts. Posit the base fact $P \rightarrow Q$ and also the query $P \rightarrow Q$. A common way of proving the query is to assume P and derive Q by modus ponens. If modus ponens is not restricted then the retriever may not terminate but if it is restricted then $P \rightarrow Q$ may not be retrievable. Situations similar to this are pervasive since theorem provers typically prove a complex sentence by using inference to recursively construct the sentence from its constituent parts. We seek a proof technique that can respond to a query simply by noticing that it is a base fact.

2.3 Specifying Limited Inference

Our method of specifying limited inference requires a shift in viewpoint from an algorithmic description of the inference engine to a higher level functional description. This is done by focusing on the retriever's query set, Q(B), which is the set of queries that succeed given a particular set of base axioms, B. A specification of Q(B) is more appropriate than a specification of its decision procedure for certain purposes, such as proving that the retrieval system has properties like the ones we are about to discuss.

The previous section argued informally that a retriever must have certain properties. We now examine these more closely from the new viewpoint. The first requirement is derived from the stipulation that all retrievals must terminate. A set is recursive if and only if there is an algorithm that in finite time determines whether any given object is a member of the set. This places the following requirement on the query set:

(Req_1) Q(B) is recursive.

It is this requirement which necessitates limitations on a retriever's inference. If the retriever is a complete theorem prover, then Q(B) is not necessarily recursive.

The second of the requirements is that if a query is in the query set then it logically follows from the base axioms.

(Req_2) $Q(B) \subseteq \{q \mid B \Rightarrow q\}$

This essentially states that the KB retrieval mechanism is sound. The third and final requirement is that all base facts can be retrieved. That is, Q(B) contains B.

(Req_3) $Q(B) \supseteq B$

Our method for specifying representation language retrievability (i.e., Q(B)) is somewhat analogous to the way in which object language provability is traditionally expressed in a meta-language. Because we do not introduce a meta-language term to name each base fact, we use the expressions "representation language" and "retrieval language" rather than "object language" and "meta-language."

The specification of a retriever has two components: a mapping, M, of representation language sentences to retrieval language sentences, and a set of retrieval axioms, R. (On occasion, M will be applied to a set, in which case it designates the set derived by applying the mapping to each member.) A query, q, succeeds if and only if its retrieval language representation, M(q), logically follows from the retrieval language representation of the base axioms, M(B), and the retrieval axioms, R. The retriever does not try to decide the truth of $B \Rightarrow q$, but rather the much simpler decision of the truth of $M(B) \cup R \Rightarrow M(q)$. Retrieval is *limited* inference since the latter sequent implies the first, but not vice-versa.

It will be seen that a quite elaborate retriever can be specified clearly and succinctly with this method. This is possible because the mappings allow representation language constructs to be embedded in the retrieval language, rather than interpreted by it. For example, it would suffice to require that representation language sentences are mapped to retrieval language sentences by quotation. However, doing so, would require that a universally quantified variable be given its meaning in the retrieval language by defining operations such as *substitution*. *This is what we have referred to as interpretation*. On the other hand, the meaning of a universally quantified variable could be *embedded* in the retrieval language simply by mapping representation language variables to retrieval language variables. Interpretation is a more general technique but, when possible, embedding will be used for its simplicity.

It is crucial to observe that in our system everything is mapped to the retrieval language and it is there that retrievability is decided. For all intents and purposes, the representation language has been discarded. This is in contrast to FOL (Weyhrauch, 1980) which uses reflection rules to map back and forth to its META representation when desired and the Bowen and Kowalski (1982) proposal which allows meta-language and object language to be mixed freely.

2.4 A Logical Retrieval System

In this section we develop a logical retrieval system by specifying M and R. We call it a *logical* system because the mappings and retrieval axioms deal specifically with each logical symbol but treat all non-logical symbols uniformly. We have *deliberately* made this system quite weak because it has

no control knowledge of how to handle any specific predicate or function symbols. We strengthen the system in section 5 by dealing explicitly with a small set of predicates used to structure semantic networks.

The mapping M, which takes representation language sentences to retrieval language sentences, will be explained by demonstrating how it handles several sample formulae. A literal in the representation language maps to a term in the retrieval language by mapping predicate symbols and function symbols to corresponding function symbols and the logical operator ~ to the function symbol NOT. Thus the representation language literal ~DRINKS(JB,BEER) maps to the retrieval language term NOT(DRINKS(JB,BEER)). Since representation language and retrieval language are never mixed, we adopt the convention of mapping function symbols and predicate symbols to retrieval language function symbols of the same name. It should always be clear from context which language a given formula is in.

A representation language disjunction of literals is mapped to the retrieval language as a set of terms. For example, ~DRINKS(JB,MILK) $\vee$ DRINKS(JB,BEER) maps to the term {NOT(DRINKS(JB,MILK)),DRINKS(JB,BEER)}. By virtue of sets being *unordered* collections, the retrieval language accounts for the fact that representation language $\vee$ is commutative and associative.

Now consider how the base fact

(1) DRINKS(JB,MILK) $\rightarrow$ DRINKS(JB,BEER)

is mapped to the retrieval language. First the base fact is put into prenex conjunctive normal form, which in this case yields the single conjunct:

(2) ~DRINKS(JB,MILK) $\vee$ DRINKS(JB,BEER)

This disjunction of literals is then mapped to a retrieval language term that is then asserted to be retrievable with the predicate RET. Thus (2) is mapped to the atomic sentence

(3) RET({NOT(DRINKS(JB,MILK)),DRINKS(JB,BEER)})

If the sentence to be mapped has more than one conjunct, such as

(4) DRINKS(ALAN,MILK) $\wedge$ DRINKS(ALAN,BEER)

then each conjunct is mapped as before. The resulting atomic sentences are then made into a conjunction. Thus, mapping (4) would result in the retrieval language sentence

(5) RET({DRINKS(ALAN,MILK)}) ∧ RET({DRINKS(ALAN,BEER)})

It can be seen that object language conjunction is mapped to retrieval language conjunction. This method of defining the meaning of a representation language construct by mapping it directly to its corresponding retrieval language construct is what we have referred to as embedding.

Object language quantification is also embedded in the retrieval language. Consider the sentence

(6) ∀x (∃y DRINKS(x,y)) ∧ (∃y ~DRINKS(x,y))

which in prenex conjunctive normal form becomes

(7) ∀x ∃y ∃z DRINKS(x,y) ∧ ~DRINKS(x,z)

The matrix is mapped as before with the additional consideration that variables are mapped to corresponding variables. The prefix remains essentially unchanged by the mapping except for the mapping of variables to corresponding variables. Thus, mapping (7) to the retrieval language results in

(8) ∀x ∃y ∃z RET({DRINKS(x,y)}) ∧ RET({NOT(DRINKS(x,z))})

Observe that since the representation language connectives ∨ and ~ have been mapped to retrieval language function symbols, their meaning will be interpreted by the retrieval language. There will be axioms which enable deductions to be made with the terms constructed from NOT and { }. These will mimic sound representation language inferences. However, the representation language proof system must not be totally mimicked by the retrieval language axioms if (Req_1) is to be met. OR and NOT will be given much weaker interpretations than ∨ and ~. On the other hand, representation language ∧ and quantification has been mapped to retrieval language ∧ and quantification. This embedding gives these representation language constructs their full-blown logical meaning.

Before considering retrieval axioms, let us examine this retriever with no

retrieval axioms. It can immediately be seen that this retriever meets (Req_3). It also meets the first two requirements, but showing so involves more subtlety than can be dealt with in this paper.

But the query set contains much more than just the base axioms! For example, the query

(9) $\sim\exists x\ (\forall y\ DRINKS(x,y)) \vee (\forall y \sim DRINKS(x,y))$

would succeed in a KB that had (6) as a base fact. Though not immediately obvious, (6) and (9) both map to (8) in the retrieval language and are therefore logically equivalent. The normalization process that accounts for this by equating many logical paraphrases is a powerful technique. Mapping disjunctions to sets is also a form of normalization because it makes the retrieval language insensitive to the ordering of representation language disjunction. Thus, if $A \vee B$ were added to the KB, the query $B \vee A$ would succeed.

In addition, the embedding of representation language quantification and conjunction means that these will be handled with their full logical meaning. For example, the query

(10) $\exists x\ \exists y\ DRINKS(x,y)$

can be answered positively by this KB if either (4) or (7) had been added. The retrieval language representation of this query,

(11) $\exists x\ \exists y\ RET(\{DRINKS(x,y)\})$

logically follows from (5), the retrieval language representation of (4), or from (8), the retrieval language representation of (7).

Because there are no retrieval axioms that chain base facts together (i.e., use more than one base fact in an inference) this system has the property that

($Prop_1$) A query succeeds *only if* each conjunct of the query follows from one conjunct of one base fact.

We will now look at ways of strengthening this system by adding retrieval axioms. These retrieval axioms strengthen the system but not so much as to destroy ($Prop_1$). The first retrieval axiom is based on the fact that disjunction is

monotonic. That is, if a disjunction is true, then adding additional disjuncts to it cannot change its truth value. This inspires the logical retrieval axiom:

(R_1) $\quad \forall y\,(\exists x\,\mathrm{RET}(x) \wedge x \subseteq y) \rightarrow \mathrm{RET}(y)$

The above retrieval axiom allows a set to be manipulated in a manner consistent with the meaning of the disjunction it encodes. The next retrieval axiom does the same for NOT and the negation it encodes. It says that an atomic sentence and its negation cannot both be retrievable.

(R_2) $\quad \forall x\ \sim\mathrm{RET}(\{x\}) \vee \sim\mathrm{RET}(\{\mathrm{NOT}(x)\})$

(R_2) does not change the query set of the current retriever but it is needed in the semantic network retrieval system.

Because (Prop_1) is a desirable design objective, (R_1) and (R_2) are the only logical retrieval axioms used. There will be other retrieval axioms, such as those derived from the study of semantic networks undertaken in the next section. Although we consider the three requirements which we presented to be constraints on all KB designs, the design objective was chosen based on our particular task. We shall now discuss the rationale for this design objective.

As discussed before, and as can be seen clearly now, the KB has no special knowledge of any representation language predicates. It therefore treats them uniformly and this is what we mean when we say that the KB is domain independent. Without such knowledge, a strong inferencial component cannot perform effectively and thus we have taken a conservative approach to inference. In this sense the system presented here can be viewed as a KB kernel on top of which a more powerful, and possibly domain-dependent KB could be built. A reasoner can conpensate for a KB that is efficient but deductively weak by making deductions of its own. However, it is hard for a reasoner to compensate for a KB whose excessive deductive power leads to inefficiency.

2.5 A Semantic Network Retrieval System

In this section, we investigate a very simple semantic network system by formalizing it in FOPC. This investigation yields a primitive set of predicates tha

can be used to structure knowledge and a set of axioms relating these predicates. This formalization takes place without consideration of the role that the axioms should play in a KB. We then take up the problem of how to integrate these axioms into the KB so that the retriever can use them to make the kind of inferences typically performed by semantic network interpreters.

The fact that our KB uses the notation of FOPC for the representation language is methodologically important yet it says little about how a domain should be represented. Modern semantic networks - those since (Woods, 1975) - have made a steps in suggesting knowledge structuring primitives, though a great deal of freedom still remains in choosing how to represent a domain. Others have referred to these knowledge-structuring primitives as epistemological primitives (Brachman, 1979), structural relations (Shapiro, 1979) and system relations (Shapiro, 1971).

Elsewhere (Allen and Frisch, 1982), we have shown how semantic networks can motivate a logic with a fixed set of predicates and how the relationship between these predicates can be axiomatized. Here we use a much simpler scheme that has 4 predicates, TYPE, SUBTYPE, ROLE, and =, and 3 axioms, labelled (Asn-1), (Asn-2), and (Asn-3). We then integrate these axioms into the KB so that they can drive the retriever.

The use of type hierarchies is characteristic of semantic networks. The domain of individuals is divided into sets called types. TYPE(I,T) asserts that an individual I belongs to a type T and SUBTYPE(T,T') asserts that the type T is a subset of the type T'. There are two axioms involving these predicates:

(Asn_1) $\forall x,t,t'\ \mathrm{TYPE}(x,t') \wedge \mathrm{SUBTYPE}(t',t) \rightarrow \mathrm{TYPE}(x,t)$

(Asn_2) $\forall x,t',t\ \mathrm{SUBTYPE}(t',x) \wedge \mathrm{SUBTYPE}(x,t) \rightarrow \mathrm{SUBTYPE}(t',t)$

If events such as 'Jellybean drank milk' are represented as DRANK(JB,MILK) then, as pointed out by Davidson (1967), there is no way to quantify over events and their components. This prevents the representation of assertions such as 'The actor of an action causes that action.' For this reason and for the purpose of making all relations binary, semantic networks traditionally represent 'Jellybean drank milk' as

(12) TYPE(DRANK01,DRANKEVENT) ∧ ACTOR(DRANK01,JB) ∧
OBJECT(DRANK01,MILK)

Thus, 'The actor of an event causes that event' can be expressed as

(13) $\forall x,y\ \text{TYPE}(x,\text{ACTION}) \wedge \text{ACTOR}(x,y) \rightarrow \text{CAUSE}(y,x)$

However, in this representation, there is no way to state 'Role fillers are unique' or to query 'Is there an event involving Jellybean and milk.' Because we do not restrict ourselves to binary relations, we can generalize the above representation by making roles into objects in their own right. Thus, (12) becomes

(14) $\text{TYPE}(\text{DRANK01},\text{DRANKEVENT}) \wedge \text{ROLE}(\text{ACTOR},\text{DRANK01},\text{JB}) \wedge \text{ROLE}(\text{OBJECT},\text{DRANK01},\text{MILK})$

and the query 'Is there an event involving Jellybean and milk' is represented as:

(15) $\exists e,r,r'\ \text{TYPE}(e,\text{EVENT}) \wedge \text{ROLE}(r,e,\text{JB}) \wedge \text{ROLE}(r',e,\text{MILK})$

Notice that (15) logically follows from (14).

The third and final semantic network axiom states that role fillers are unique and now can be expressed as:

(Asn_3) $\forall r,e,f,f'\ \text{ROLE}(r,e,f) \wedge \text{ROLE}(r,e,f') \rightarrow f = f'$

How can (Asn_1), (Asn_2) and (Asn_3) be integrated into the KB so that they can be used by the retriever in its deductions? We would like to obtain the property that the query set is closed with respect to the derivation of true sentences from these axioms.

First of all, these axioms should be added to the set of base facts in the KB. Mapping them to the retrieval language yields:

(16) $\forall x,t,t'\ \text{RET}(\{\text{NOT}(\text{TYPE}(x,t')),\text{NOT}(\text{SUBTYPE}(t',t)),\text{TYPE}(x,t)\})$

(17) $\forall x,t',t\ \text{RET}(\{\text{NOT}(\text{SUBTYPE}(t',x)),\text{NOT}(\text{SUBTYPE}(x,t)),\text{SUBTYPE}(t',t)\})$

(18) $\forall r,e,f,f'\ \text{RET}(\{\text{NOT}(\text{ROLE}(r,e,f)),\text{NOT}(\text{ROLE}(r,e,f')),f = f'\})$

But this is not enough. Let us consider the situation in which TYPE(JB,DOG) and SUBTYPE(DOG,MAMMAL) are also base facts and the query that we wish to succeed is TYPE(JB,MAMMAL). Thus, in the retrieval language there are two base facts,

(19) RET({TYPE(JB,DOG)})

(20) RET({SUBTYPE(DOG,MAMMAL)})

and the query

(21) RET({TYPE(JB,MAMMAL)})

But notice that (21) does not follow from the base facts. The query fails because disjunction in (Asn_1) is weakened when mapped to the retrieval logic. We can get the query to succeed by using the following retrieval axiom based on (Asn_1):

(R_3) $\forall x,t,t'$ RET({NOT(TYPE(x,t'))}) $\vee$ RET({NOT(SUBTYPE(t',t))}) $\vee$
RET({TYPE(x,t)})

(21) now logically follows from (19), (20), (R_2) and (R_3). The increased power of the retriever is due to the fact that the disjunction in (Asn_1) has been mapped to a disjunction in (R_3) rather than a set as was done in (16).

Likewise, the two other semantic network axioms are made into retrieval axioms:

(R_4) $\forall x,t',t$ RET({NOT(SUBTYPE(t',x))}) $\vee$ RET({NOT(SUBTYPE(x,t))}) $\vee$
RET({SUBTYPE(t',t)})

(R_5) $\forall r,e,f,f'$ RET({NOT(ROLE(r,e,f))}) $\vee$ RET({NOT(ROLE(r,e,f'))}) $\vee$
RET({f = f'})

However, a deficiency still remains. Consider the situation in which 'All mammals drink a liquid' is added to the above KB and the query 'Jellybean drinks a liquid' is made. Adding

(22) $\forall m$ TYPE(m,MAMMAL) $\rightarrow$ $\exists l$ TYPE(l,LIQUID) $\wedge$ DRINKS(m,l)

yields the retrieval language base fact

(23) $\forall m\ \exists l$ RET({NOT(TYPE(m,MAMMAL)),TYPE(l,LIQUID)}) $\wedge$
RET{NOT(TYPE(m,MAMMAL)),DRINKS(m,l)})

Though (21) logically follows, (23) is not strong enough to logically imply the queried sentence.

(24) $\exists l\ \mathrm{RET}(\{\mathrm{TYPE}(l,\mathrm{LIQUID})\}) \wedge \mathrm{RET}(\{\mathrm{DRINKS}(\mathrm{JB},l)\})$

More generally, the KB as it now stands cannot infer a property of an individual based on a property asserted of all members of one of its types. The problem is that M combines TYPE atoms with other atoms thus subjecting types to the system's inference limitations. The problem could be solved by altering M to factor out appropriate representation language TYPE atoms into their own retrieval language RET atoms. For example, the query (24) would succeed if (22) were mapped to the retrieval language as

(25) $\forall m\ \exists l\ \mathrm{RET}(\{\mathrm{TYPE}(m,\mathrm{MAMMAL})\}) \rightarrow$
$\mathrm{RET}(\{\mathrm{TYPE}(l,\mathrm{LIQUID})\}) \wedge \mathrm{RET}(\{\mathrm{DRINKS}(m,l)\})$

Rather than complicating the mappings to recognize and factor out the appropriate type information, the representation language is extended so that certain type information can be written in a factored-out manner initially. The extended language, called typed first-order predicate calculus or TFOPC, includes formulae of the form $\forall x{:}\tau\ \varphi$ and $\exists x{:}\tau\ \varphi$, where φ is a formula and τ is the name of a type. The meaning of this notation is defined by two equivalence schemata:

(26) $\forall x{:}\tau\ \varphi \equiv \forall x\ \mathrm{TYPE}(x,\tau) \rightarrow \varphi$

(27) $\exists x{:}\tau\ \varphi \equiv \exists x\ \mathrm{TYPE}(x,\tau) \wedge \varphi$

Thus, (22) could be written in TFOPC as

(28) $\forall m{:}\mathrm{MAMMAL}\ \exists l{:}\mathrm{LIQUID}\ \mathrm{DRINKS}(m,l)$

M is appropriately modified to handle the extended representation language. As before, the representation language base fact or query is put in prenex conjunctive normal form. The same process that converts a FOPC sentence to prenex conjunctive normal form (Robinson, 1979) also converts a TFOPC sentence. Here's why: The conversion process is based on many equivalence schemata, six of which deal with quantification. There are six corresponding TFOPC schemata that can easily be proved and therefore, quantifiers can be moved about as if the types weren't even there.

Once the query or base fact is in prenex conjunctive normal form, it is mapped as before but with the additional consideration that formulae of the form $\forall x{:}\tau\ \varphi$ map to

(29) $\forall x\ RET(\{TYPE(x,\tau)\}) \rightarrow M(\varphi)$

and those of the form $\exists x{:}\tau\ \varphi$ map to

(30) $\exists x\ RET(\{TYPE(x,\tau)\}) \wedge M(\varphi)$

Therefore, applying the complete M mapping to the base fact (28) yields

(31) $\forall m\ RET(\{TYPE(m,MAMMAL)\}) \rightarrow$
$\exists l\ RET(\{TYPE(l,LIQUID)\}) \wedge RET(\{DRINKS(m,l)\})$

The query 'Jellybean drinks a liquid,' which can be stated in TFOPC as

(32) $\exists l{:}LIQUID\ DRINKS(JB,l)$

gets mapped to the retrieval language as

(33) $\exists l\ RET(\{TYPE(l,LIQUID)\}) \wedge RET(\{DRINKS(JB,l)\})$

And finally, the deficiency stated at the outset has been overcome. The example situation now behaves as desired since (33) logically follows from (19), (20), (31), and the retrieval axioms.

As a final observation, notice that typing existential variables has not extended the power of the system as has typing universal variables. This is due to the fact that a type on an existential variable is an abbreviation for a conjunction and the limited inference handles conjunction fully. Typed existential variables have been added to the language for the sake of uniformity.

The semantic-network retriever presented in this section meets (Req_1), (Req_2) and (Req_3). Though the retriever can now chain base facts together in deriving TYPE and SUBTYPE relations, Q(B) is still recursive because the type hierarchy is finite.

2.6 Implementing Retrieval

The KB module of the Rochester Dialog System has a retriever that is an extended version of the semantic-network retriever presented here. The representation language has been extended with a set of abbreviations tailored to our domain and the semantic network retrieval axioms have been extended to handle a larger set of knowledge-structuring primitives (Allen and Frisch, 1982). All communication with the KB is in the representation language; the

retrieval language is totally invisible. We will only briefly discuss the techniques used in the implementation since we have not yet proved their correctness.

Our method of producing a program that meets the specification developed in this paper is to map queries and base facts to the retrieval language and treat the specification as the logic component of an algorithm (Kowalski, 1979). This logic component can then be transformed to an equivalent specification, one for which we can produce an efficient control component. The transform we employ makes all retrieval language sentences into Horn clauses. Notice that all of the semantic-network retrieval axioms are disjunctions of positive RET literals of singleton sets. For concreteness consider

(R_3) $\forall x,t,t'$ RET({NOT(TYPE(x,t'))}) ∨ RET({NOT(SUBTYPE(t',t))}) ∨ RET({TYPE(x,t)})

There are three sentences that have a single positive literal which logically follow from (ASN_1) and

(R_2) $\forall x$ ~RET({x}) ∨ ~RET({NOT(x)})

They are:

(34) $\forall x,t,t'$ RET({NOT(TYPE(x,t'))}) ∨ ~RET({SUBTYPE(t',t)}) ∨ ~RET({NOT(TYPE(x,t))})

(35) $\forall x,t,t'$ ~RET({TYPE(x,t')}) ∨ RET({NOT(SUBTYPE(t',t))}) ∨ ~RET({NOT(TYPE(x,t))})

(36) $\forall x,t,t'$ ~RET({TYPE(x,t')}) ∨ ~RET({SUBTYPE(t',t)}) ∨ RET({TYPE(x,t)})

Thus every sentence of *n* literals is rewritten into *n* sentences, each of which has one positive literal and *n-1* negative literals. Once this rewriting is completed (R_2) is no longer needed. This rewriting is clearly sound and, in general, incomplete. The important question remains open: Is this rewriting technique complete for our class of theories? The new set of sentences does not need to be equivalent to the original set of sentences; only the truth of the sequent M(B) ∪ R ⇒ M(q) needs to be preserved. We point out that this rewriting is related to Meltzer's proposal (1966) and appears to yield a system

equivalent to linear input resolution (Loveland, 1978).

It can be seen that these rewritten sentences can then trivially be made into Horn clauses. We then use a PROLOG-like theorem prover called HORNE (Allen and Frisch, 1981; Frisch, Allen and Giuliano, 1982) to interpret the clauses. Failure to prove a query is construed as meaning that the query is not in Q(B) (Clark, 1978). We now discuss the selection and search strategies (Van Emden, 1977) that we use to control the theorem-prover.

Since all clauses that result from retrieval axioms are fixed, we manually order the literals within the clauses to take advantadge of HORNE's left-to-right selection strategy. As discussed by Clark and McCabe (1979), several orderings are specified for each clause depending on which variables are bound. Cuts are also manually added to these clauses based on the bindings. Ordering of literals within base facts is not crucial since they are atomic with the exception of the TYPE literals. Ordering of literals within a query is crucial and only can be done at run-time. Nothing has been done in this regard but some thought has been given to using a method similar to Chat-80's query optimization (Warren and Pereira, 1981).

Search strategy plays a relatively minor role in the retriever's performance. When the retriever is looking for all answers to a query or when there are no answers to a query, the entire search space must be examined. These situations are not unusual and clearly search strategy is irrelevant in them. Therefore, HORNE's depth-first search is a good choice because it lends itself to efficient implementation through the use of stack allocation. The order that branches are chosen is arbitrary.

However, if our search space were dynamically pruned, search strategy would be a factor. Not only could dynamic pruning eliminate deductive paths guaranteed to fail (Perreira and Porto. 1980) but it could eliminate deductive paths guaranteed to yield redundant solutions.

We have realized great improvements in efficiency by using TFOPC as the retrieval language and accordingly extending HORNE to handle TFOPC Horn clauses. Type checking is done on unification by recursively calling the retriever to test the appropriate TYPE and SUBTYPE relations. This organization reorders the goals in a proof and appears to result in a much smaller search

space. The rationale for this is that, when possible, the retriever reasons about classes of individuals rather than about the individuals themselves. This is a minimum commitment strategy similar to that obtained in MOLGEN (Stefik, 1981) by the use of constraints.

Chapter 3

What's in a Semantic Network?

Abstract

Ever since Woods's "What's in a Link" paper, there has been a growing concern for formalization in the study of knowledge representation. Several arguments have been made that frame representation languages and semantic-network languages are syntactic variants of the first-order predicate calculus (FOPC). The typical argument proceeds by showing how any given frame or network representation can be mapped to a logically isomorphic FOPC representation. For the past two years we have been studying the formalization of knowledge retrievers as well as the representation languages that they operate on. This paper presents a representation language in the notation of FOPC whose form facilitates the design of a semantic-network-like retriever.

3.1 Introduction

We are engaged in a long-term project to construct a system that can partake in extended English dialogues on some reasonably well specified range of topics. A major part of this effort so far has been the specification of a knowledge representation. Because of the wide range of issues that we are trying to capture, which includes the representation of plans, actions, time, and individuals' beliefs and intentions, it is crucial to work within a framework general enough to accommodate each issue. Thus, we began developing our representation within the first-order predicate calculus. So far, this has presented no problems, and we aim to continue within this framework until some problem forces us to do otherwise.

Given this framework, we need to be able to build reasonably efficient systems for use in the project. In particular, the knowledge representation must be able to support the natural language understanding task. This requires that certain forms of inference must be made. Within a general theorem-proving framework, however, those inferences desired would be lost within a wide range of undesired inferences. Thus we have spent considerable effort in constructing a specialized inference component that can support the language

understanding task.

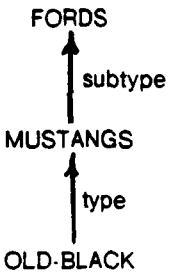
Before such a component could be built, we needed to identify what inferences were desired. Not surprisingly, much of the behavior we desire can be found within existing semantic network systems used for natural language understanding. Thus the question "What inferences do we need?" can be answered by answering the question "What's in a semantic network?"

Ever since Woods's (1975) "What's in a Link" paper, there has been a growing concern for formalization in the study of knowledge representation. Several arguments have been made that frame representation languages and semantic-network languages are syntactic variants of the first-order predicate calculus (FOPC). The typical argument (e.g., (Hayes, 1979; Nilsson, 1980; Charniak, 1981a)) proceeds by showing how any given frame or network representation can be mapped to a *logically isomorphic* (i.e., logically equivalent when the mapping between the two notations is accounted for) FOPC representation. We emphasize the term "logically isomorphic" because these arguments have primarily dealt with the content (semantics) of the representations rather than their forms (syntax). Though these arguments are valid and scientifically important, they do not answer our question.

Semantic networks not only represent information but facilitate the retrieval of relevant facts. For instance, all the facts about John are stored with a pointer directly to one node representing John (e.g., see the papers in (Findler, 1979)). Another example concerns the inheritance of properties. Given a fact such as "All canaries are yellow," most network systems would automatically conclude that "Tweety is yellow," given that Tweety is a canary. This is typically implemented within the network matcher or retriever.

We have demonstrated elsewhere (Frisch and Allen, 1982) the utility of viewing a knowledge retriever as a specialized inference engine (theorem prover). A specialized inference engine is tailored to treat certain predicate, function, and constant symbols differently than others. This is done by building into the inference engine certain true sentences involving these symbols and the control needed to handle with these sentences. The inference engine must also be able to recognize when it is able to use its specialized machinery. That is, its specialized knowledge must be coupled to the *form* of the situations that it can deal with.

For illustration, consider an instance of the ubiquitous type hierarchies of semantic networks:



By mapping the types AUTOS and MUSTANGS to be predicates which are true only of automobiles and mustangs respectively, the following two FOPC sentences are logically isomorphic to the network:

(1.1) $\forall x$ MUSTANGS(x) $\rightarrow$ FORDS(x)

(1.2) MUSTANGS(OLD-BLACK1)

However, these two sentences have not captured the form of the network, and furthermore, not doing so is problematic to the design of a retriever. The subtype and type links have been built into the network language because the network retriever has been built to handle them specially. That is, the retriever does not view a subtype link as an arbitrary implication such as (1.1) and it does not view a type link as an arbitrary atomic sentence such as (1.2).

In our representation language we capture the form as well as the content of the network. By introducing two predicates, TYPE and SUBTYPE, we capture the meaning of the type and subtype links. TYPE(i,t) is true iff the individual i is a member of the type (set of objects) t, and SUBTYPE(t_1,t_2) is true iff the type t_1 is a subtype (subset) of the type t_2. Thus, in our language, the following two sentences would be used to represent what was intended by the network:

(2.1) SUBTYPE(FORDS,MUSTANGS)

(2.2) TYPE(OLD-BLACK1,FORDS)

It is now easy to build a retriever that recognizes subtype and type assertions by matching predicate names. Contrast this to the case where the representation language used (1.1) and (1.2) and the retriever would have to

recognize these as sentences to be handled in a special manner.

But what must the retriever know about the SUBTYPE and TYPE predicates in order that it can reason (make inferences) with them? There are two assertions, (A.1) and (A.2), such that {(1.1),(1.2)} is logically isomorphic to {(2.1),(2.2),(A.1),(A.2)}. (Note: throughout this paper, axioms that define the retriever's capabilities will be referred to as *built-in axioms* and specially labeled A.1, A.2, etc.)

(A.1) $\forall t_1,t_2,t_3\ \text{SUBTYPE}(t_1,t_2) \wedge \text{SUBTYPE}(t_2,t_3) \rightarrow \text{SUBTYPE}(t_1,t_3)$
(SUBTYPE is transitive.)

(A.2) $\forall o,t_1,t_2\ \text{TYPE}(o,t_1) \wedge \text{SUBTYPE}(t_1,t_2) \rightarrow \text{TYPE}(o,t_2)$
(Every member of a given type is a member of its supertypes.)

The retriever will also need to know how to control inferences with these axioms, but this issue is considered only briefly in this paper.

The design of a semantic-network language often continues by introducing new kinds of nodes and links into the language. This process may terminate with a fixed set of node and link types that are the knowledge-structuring primitives out of which all representations are built. Others have referred to these knowledge-structuring primitives as epistemological primitives (Brachman, 1979), structural relations (Shapiro, 1979), and system relations (Shapiro, 1971). If a fixed set of knowledge-structuring primitives is used in the language, then a retriever can be built that knows how to deal with all of them.

The design of our representation language very much mimics this approach. Our knowledge-structuring primitives include a fixed set of predicate names and terms denoting three kinds of elements in the domain. We give meaning to these primitives by writing domain-independent axioms involving them. Thus far in this paper we have introduced two predicates (TYPE and SUBTYPE), two kinds of elements (individuals and types), and two axioms ((A.1) and (A.2)). We shall name types in uppercase and individuals in uppercase letters followed by at least one digit.

Considering the above analysis, a retrieval now is viewed as an attempt to prove some queried fact logically follows from the *base facts* (e.g., (2.1), (2.2)) and the built-in axioms (such as (A.1) and (A.2)). For the purposes of this paper,

we can consider all base facts to be atomic formulae (i.e., they contain no logical operators except negation). While compound formulae such as disjunctions can be represented, they are of little use to the semantic network retrieval facility, and so will not be considered in this paper. We have implemented a retriever along these lines and it is currently being used in the Rochester Dialogue System (Allen, Frisch and Litman, 1982).

3.2 The Basic Representation: Objects, Events, and Relations

An important property of a natural language system is that it often has only partial information about the individuals (objects, events, and relations) that are talked about. Unless one assumes that the original linguistic analysis can resolve all these uncertainties and ambiguities, one needs to be able to represent partial knowledge. Furthermore, the things talked about do not necessarily correspond to the world: objects are described that don't exist, and events are described that do not occur.

In order to be able to capture such issues we will need to include in the domain all conceivable individuals (cf. all conceivable concepts (Brachman, 1979)). We will then need predicates that describe how these concepts correspond to reality. The class of individuals in the world is subcategorized into three major classes: objects, events, and relations. We consider each in turn.

3.2.1 Objects

Objects include all conceivable physical objects as well as abstract objects such as ideas, numbers, etc. The most important knowledge about any object is its type. Mechanisms for capturing this were outlined above. Properties of objects are inherited from statements involving universal quantification over the members of a type. The fact that a physical object, o, actually exists in the world will be asserted as IS-REAL(o).

3.2.2 Events

The problems inherent in representing events and actions are well described by Davidson (1967). He proposes introducing events as elements in the domain and introducing predicates that modify an event description by adding a role

(e.g., agent, object) or by modifying the manner in which the event occurred. The same approach has been used in virtually all semantic network- and frame-based systems (Charniak, 1981b), most of which use a case grammar (Fillmore, 1968) to influence the choice of role names. This approach also enables quantification over events and their components such as in the sentence, "For each event, the actor of the event causes that event." Thus, rather than representing the assertion that the ball fell by a sentence such as

(try-1) FALL(BALL1).

the more appropriate form is

(try-2) ∃e TYPE(e,FALL-EVENTS) ∧ OBJECT-ROLE(e,BALL1).

This formalism, however, does not allow us to make assertions about roles in general, or to assert that an object plays some role in an event. For example, there is no way to express "Role fillers are unique" or "There is an event in which John played a role." Because we do not restrict ourselves to binary relations, we can generalize our representation by introducing the predicate ROLE and making rolenames into individuals in the domain. ROLE(o,*r*,v) asserts that individual o has a role named *r* that is filled with individual v. To distinguish rolenames from types and individuals, we shall use italics for rolenames.

Finally, so that we can discuss events that did not occur (as opposed to saying that such an event doesn't exist), we need to add the predicate OCCURS. OCCURS(e) asserts that event e actually occurred. Thus, finally, the assertion that the ball fell is expressed as

(3) ∃e TYPE(e,FALL-EVENTS) ∧ ROLE(e,*OBJECT*,BALL1) ∧ OCCURS(e).

Roles are associated with an event type by asserting that every individual of that type has the desired role. To assert that every event has an *OBJECT* role, we state

(4) ∀e ∃v TYPE(e,EVENTS) → ROLE(e,*OBJECT*,v).

Given this formulation, we could now represent that "some event occurred involving John" by

(5) ∃e,*rolename* TYPE(e.EVENTS) ∧ ROLE(e,*rolename*,JOHN1) ∧ OCCURS(e)

By querying fact (5) in our retriever, we can find all events involving John.

One of the most important aspects of roles is that they are functional, e.g., each event has exactly one object role, etc. Since this is important in designing an efficient retriever, it is introduced as a built-in axiom:

(A.3) $\forall r,o,v_1,v_2\ \mathrm{ROLE}(o,r,v_1) \wedge \mathrm{ROLE}(o,r,v_2) \rightarrow (v_1 = v_2)$.

3.2.3 Relations

The final major type that needs discussing is the class of relations. The same problems that arise in representing events arise in representing relations. For instance, often the analysis of a simple noun-noun phrase such as "the book cook" initially may be only understood to the extent that some relationship holds between "book" and "cook." If we want to represent this, we need to be able to partially describe relations. This problem is addressed in semantic networks by describing relations along the same lines as events.

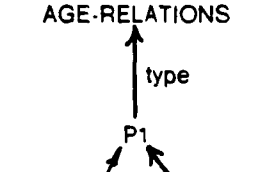
For example, rather than expressing "John is 10" as

(6) AGE-OF(JOHN1,10)

we use the TYPE and ROLE predicates introduced above to get

(7) $\exists p$ TYPE(p,AGE-RELATIONS) $\wedge$ ROLE(p,*OBJECT*,JOHN1) $\wedge$
ROLE(p,*VALUE*,10).

This, of course, mirrors a semantic network such as



As with events, describing a relation should not entail that the relation holds. If this were the case, it would be difficult to represent non-atomic sentences such as a disjunction, since in describing one of the disjuncts, we would be asserting that the disjunct holds. We assert that a relation, r, is true with

HOLDS(r). Thus the assertion that "John is 10" would involve (7) conjoined with HOLDS(p), i.e.,

(8) $\exists p$ TYPE(p,AGE-RELATIONS) $\wedge$ ROLE(p.*OBJECT*,JOHN1) $\wedge$ ROLE(p.*VALUE*.10) $\wedge$ HOLDS(p)

The assertion "John is not 10" is not the negation of (8), but is (7) conjoined with ~HOLDS(p), i.e.,

(9) $\exists p$ TYPE(p.AGE-RELATIONS) $\wedge$ ROLE(p.*OBJECT*,JOHN1) $\wedge$ ROLE(p,*VALUE*,10) $\wedge$ ~HOLDS(p).

We could also handle negation by introducing the type NOT-RELATIONS, which takes one role that is filled by another relation. To assert the above, we would construct an individual N1, of type NOT-RELATIONS, with its role filled with p, and assert that N1 holds. We see no advantage to this approach, however, since negation "moves through" the HOLDS predicate. In other words, the relation "not p" holding is equivalent to the relation "p" not holding. Disjunction and conjunction are treated in a similar manner.

3.3 Making Types Work for You

The system described so far, though simple, is close to providing us with one of the most characteristic inferences made by semantic networks, namely inheritance. For example, we might have the following sort of information in our network:

(10) SUBTYPE(MAMMALS,ANIMALS)

(11) SUBTYPE(2-LEGGED-ANIMALS,ANIMALS)

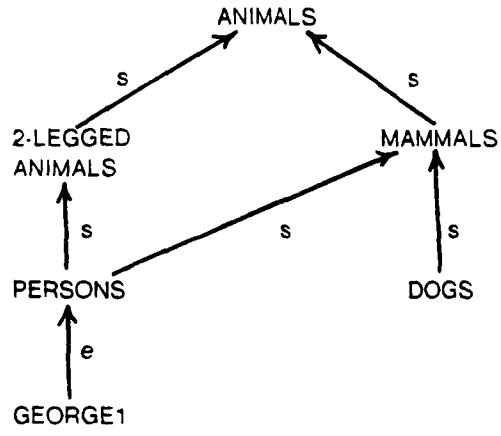
(12) SUBTYPE(PERSONS,MAMMALS)

(13) SUBTYPE(PERSONS.2-LEGGED-ANIMALS)

(14) SUBTYPE(DOGS,MAMMALS)

(15) TYPE(GEORGE1,PERSONS)

In a notation similar to that used by Hendrix (1979), these facts would be represented as:



In addition, let us assume we know that all instances of 2-LEGGED-ANIMALS have two legs and that all instances of MAMMALS are warm-blooded:

(16) $\forall x$ TYPE(x,2-LEGGED-ANIMALS) → HAS-2-LEGS(x)

(17) $\forall y$ TYPE(y,MAMMALS) → WARM-BLOODED(y)

These would be captured in the Hendrix formalism using his delineation mechanism.

Note that relations such as WARM-BLOODED and HAS-2-LEGS should themselves be described as relations with roles, but that is not necessary for this example. Given these facts, and axioms (A.1) to (A.3), we can prove that "George has two legs" by using axiom (A.2) on (13) and (15) to conclude

(18) TYPE(GEORGE1,2-LEGGED-ANIMALS)

and then using (18) with (16) to conclude

(19) HAS-2-LEGS(GEORGE1).

In order to build a retriever that can perform these inferences automatically, we must be able to distinguish facts like (16) and (17) from arbitrary facts involving implications, for we cannot allow arbitrary chaining and retain efficiency. This could be done by checking for implications where the antecedent is composed entirely of type restrictions, but this is difficult to specify. The route we take follows the same technique described above when we introduced the TYPE and SUBTYPE predicates. We introduce new notation into the language that

explicitly captures these cases. The new form is simply a version of the typed FOPC, where variables may be restricted by the type they range over. Thus, (16) and (17) become

(20) ∀x:2-LEGGED-ANIMALS HAS-2-LEGS(x)

(21) ∀y:MAMMALS WARM-BLOODED(y)

The retriever now can be implemented as a typed theorem prover that operates only on atomic base facts (now including (20) and (21)) and axioms (A.1) to (A.3).

We now can deduce that GEORGE1 has two legs and that he is warm-blooded. Note that objects can be of many different types as well as types being subtypes of different types. Thus, we could have done the above without the type PERSONS, by making GEORGE1 of type 2-LEGGED-ANIMALS and MAMMALS.

3.4 Making Roles Work for You

In the previous section we saw how properties could be inherited. This inheritance applies to role assertions as well. For example, given a type EVENTS that has an *OBJECT* role. i.e.,

(22) SUBTYPE(EVENTS,INDIVIDUALS)

(23) ∀x:EVENTS ∃y:PHYS-OBJS ROLE(x,*OBJECT*,y).

Then if ACTIONS are a subtype of events, i.e.,

(24) SUBTYPE(ACTIONS,EVENTS).

it follows from (A.2), (23), and (24) that for every action there is something that fills its *OBJECT* role, i.e..

(25) ∀x:ACTIONS ∃y:PHYS-OBJS ROLE(x,*OBJECT*,y).

Note that the definition of the type ACTIONS could further specify the type of the values of its *OBJECT* role, but it could not contradict fact (25). Thus

(26) ∀x:ACTIONS ∃y:PERSONS ROLE(x,*OBJECT*,y).

further restricts the value of the *OBJECT* role for all individuals of type ACTIONS

to be of type PERSONS.

Another common technique used in semantic network systems is to introduce more specific types of a given type by specifying one (or more) of the role values. For instance, one might introduce a subtype of ACTION called ACTION-BY-JACK, i.e.,

(27) SUBTYPE(ACTION-BY-JACK,ACTIONS)

(28) ∀abj:ACTION-BY-JACK ROLE(abj,*ACTOR*,JACK).

Then we could encode the general fact that all actions by Jack are violent by something like

(29) ∀abj:ACTION-BY-JACK VIOLENT(abj).

This is possible in our logic, but there is a more flexible and convenient way of capturing such information. Fact (29), given (27) and (28), is equivalent to

(30) ∀a:ACTIONS ROLE(a,*ACTOR*,JACK) → VIOLENT(a).

If we can put this into a form that is recognizable to the retriever, then we could assert such facts directly without having to introduce arbitrary new types.

The extension we make this time is from what we called a type logic to a role logic. This allows quantified variables to be restricted by role values as well as type. Thus, in this new notation, (30) would be expressed as

(31) ∀a:ACTIONS [*ACTOR* JACK] VIOLENT(a).

In general, a formula of the form

$$\forall\ a{:}T\ [R_1\ V_1] \ldots [R_n\ V_n]\ P(a)$$

is logically equivalent to

$$\forall a\ (TYPE(a,T) \wedge ROLE(a,R_1,V_1) \wedge \ldots \wedge ROLE(a,R_n,V_n)) \rightarrow P(a).$$

Correspondingly, an existentially quantified formula such as

$$\exists a{:}T\ [R_1\ V_1] \ldots [R_n\ V_n]\ P(a)$$

is logically equivalent to

$$\exists a\ \text{TYPE}(a.T) \wedge \text{ROLE}(a.R_1,V_1) \wedge \ldots \wedge \text{ROLE}(a.R_n,V_n) \wedge P(a).$$

The retriever recognizes these new forms and fully reasons about the role restrictions. It is important to remember that each of these notation changes is an extension onto the original simple language. Everything that could be stated previously can still be stated. The new notation, besides often being more concise and convenient, is necessary only if the semantic network retrieval facilities are desired.

Note also that we can now define the inverse of (28), and state that all actions with actor JACK are necessarily of type ACTION-BY-JACK. This can be expressed as

(32) $\forall a$:ACTIONS [*ACTOR* JACK] TYPE(a.ACTION-BY-JACK).

3.5 Equality

One of the crucial facilities needed by natural language systems is the ability to reason about whether individuals are equal. This issue is often finessed in semantic networks by assuming that each node represents a different individual, or that every type in the type hierarchy is disjoint. This assumption has been called E-saturation by Reiter(1980b). A natural language understanding system using such a representation must decide on the referent of each description as the meaning representation is constructed, since if it creates a new individual as the referent, that individual will then be distinct from all previously known individuals. Since in actual discourse the referent of a description is not always recognized until a few sentences later, this approach lacks generality.

One approach to this problem is to introduce full reasoning about equality into the representation, but this rapidly produces a combinatorially prohibitive search space. Thus other more specialized techniques are desired. We shall consider mechanisms for proving inequality first, and then methods for proving equality.

Hendrix (1979) introduced some mechanisms that enable inequality to be proven. In his system, there are two forms of subtype links, and two forms of

instance links. This can be viewed in our system as follows: the SUBTYPE and TYPE predicates discussed above make no commitment regarding equality. However, a new relation, DSUBTYPE(t_1,t_2), asserts that t_1 is a SUBTYPE of t_2, and also that the elements of t_1 are distinct from all other elements of other DSUBTYPE's of t_2. This is captured by the axioms

(A.4) $\forall t,t_1,t_2,i_1,i_2\ (\text{DSUBTYPE}(t_1,t) \wedge \text{DSUBTYPE}(t_2,t) \wedge \text{TYPE}(i_1,t_1) \wedge \text{TYPE}(i_2,t_2) \wedge \sim\text{IDENTICAL}(t_1,t_2)) \rightarrow (i_1 \neq i_2)$

(A.5) $\forall t_1,t\ \text{DSUBTYPE}(t_1,t) \rightarrow \text{SUBTYPE}(t_1,t)$

We cannot express (A.4) in the current logic because the predicate IDENTICAL operates on the syntactic form of its arguments rather than their referents. Two terms are IDENTICAL only if they are lexically the same. To do this formally, we have to be able to refer to the syntactic form of terms. This can be done by introducing quotation into the logic along the lines of (Perlis, 1981), but is not important for the point of this paper.

A similar trick is done with elements of a single type. The predicate DTYPE(i,t) asserts that i is an instance of type t, and also is distinct from any other instances of t where the DTYPE holds. Thus we need

(A.6) $\forall i_1,i_2,t\ (\text{DTYPE}(i_1,t) \wedge \text{DTYPE}(i_2,t) \wedge \sim\text{IDENTICAL}(i_1,i_2)) \rightarrow (i_1 \neq i_2)$

(A.7) $\forall i,t\ \text{DTYPE}(i,t) \rightarrow \text{TYPE}(i,t)$

Another extremely useful categorization of objects is the partitioning of a type into a set of subtypes, i.e., each element of the type is a member of exactly one subtype. This can be defined in a similar manner as above.

Turning to methods for proving equality, Tarjan (1975) describes an efficient method for computing relations that form an equivalence class. This is adapted to support full equality reasoning on ground terms. Of course it cannot effectively handle conditional assertions of equality, but it covers many of the typical cases.

Another technique for proving equality exploits knowledge about types. Many types are such that their instances are completely defined by their roles. For such a type T, if two instances I1 and I2 of T agree on all their respective roles, then they are equal. If I1 and I2 have a role where their values are not equal,

then I1 and I2 are not equal. If we finally add the assumption that every instance of T can be characterized by its set of role values, then we can enumerate the instances of type T using a function (say t) that has an argument for each role value.

For example, consider the type AGE-RELS of age properties, which takes two roles, an *OBJECT* and a *VALUE*. Thus, the property P1 that captures the assertion "John is 10" would be described as follows:

(33) TYPE(P1,AGE-RELS) ∧ ROLE(P1,*OBJECT*,JOHN1) ∧ ROLE(P1,*VALUE*,10).

The type AGE-RELS satisfies the above properties, so any individual of type AGE-RELS with *OBJECT* role JOHN1 and *VALUE* role 10 is equal to P1. The retriever encodes such knowledge in a preprocessing stage that assigns each individual of type AGE-RELS to a canonical name. The canonical name for P1 would simply be age-rels(JOHN1,10).

Once a representation has equality, it can capture some of the distinctions made by perspectives in KRL. The same object viewed from two different perspectives is captured by two nodes, each with its own type, roles, and relations, that are asserted to be equal.

Note that one cannot expect more sophisticated reasoning about equality than the above from the retriever itself. Identifying two objects as equal is typically not a logical inference. Rather, it is a plausible inference by some specialized program such as the reference component of a natural language system which has to identify noun phrases. While the facts represented here would assist such a component in identifying possible referencts for a noun phrase given its description, it is unlikely that they would logically imply what the referent is.

3.6 Associations and Partitions

Semantic networks are useful because they structure information so that it is easy to retrieve relevant facts, or facts about certain objects. Objects are represented only once in the network, and thus there is one place where one can find all relations involving that object (by following back over incoming role arcs). While we need to be able to capture such an ability in our system, we should note that this is often not a very useful ability, for much of one's knowledge about an object will not be attached to that object but will be

acquired from the inheritance hierarchy. In a spreading activation type of framework, a considerable amount of irrelevant network will be searched before some fact high up in the type hierarchy is found. In addition, it is very seldom that one wants to be able to access all facts involving an object; it is much more likely that a subset of relations is relevant.

If desired, such associative links between objects can be simulated in our system. One could find all properties of an object O1 (including those by inheritance) by retrieving all bindings of x in the query

$$\exists x.r\ \mathrm{ROLE}(x,r,O1).$$

The ease of access provided by the links in a semantic network is effectively simulated simply by using a hashing scheme on the structure of all ROLE predicates. While the ability to hash on structures to find facts is crucial to an efficient implementation, the details are not central to our point here.

Another important form of indexing is found in Hendrix where his partition mechanism is used to provide a focus of attention for inference processes (Grosz, 1977). This is just one of the uses of partitions. Another, which we did not need, provided a facility for scoping facts within logical operators, similar to the use of parentheses in FOPC. Such a focus mechanism appears in our system as an extra argument on the main predicates (e.g., HOLDS, OCCURS, etc.).

Since contexts are introduced as a new class of objects in the language, we can quantify over them and otherwise talk about them. In particular, we can organize contexts into a lattice-like structure (corresponding to Hendrix's vistas for partitions) by introducing a transitive relation SUBCONTEXT.

(A.8) $\forall c,c1,c2\ \mathrm{SUBCONTEXT}(c,c1) \wedge \mathrm{SUBCONTEXT}(c1,c2) \rightarrow \mathrm{SUBCONTEXT}(c,c2)$

To relate contexts to the HOLDS predicate, a proposition p holds in a context c if it is known to hold in c explicitly, or it holds in a super context of c.

(A.9) $\forall p,t,c,c'\ \mathrm{SUBCONTEXT}(c,c') \wedge \mathrm{HOLDS}(p,c') \rightarrow \mathrm{HOLDS}(p,c).$

As with the SUBTYPE relation, this axiom would defy an efficient implementation if the contexts were not organized in a finite lattice structure. Of course, we

need axioms similar to (A.9) for the OCCURS and IS-REAL predicates.

3.7 Extensions

This section outlines a couple of the areas in which we are extending our system beyond the capabilities normally found in semantic networks. We have chosen to present these two areas because each illustrates an advantage to our approach.

In the first area, the context mechanism is being extended to handle time, belief contexts (based on a syntactic theory of belief (Haas, 1982)), simple hypothetical reasoning, and a representation of plans. Section 3.7.1 outlines one of these, the representation of time. This example illustrates that because the matcher is defined by a set of axioms, it is relatively simple to handle new features by adding new axioms.

The second area involves the definition of types of retrieval in addition to straightforward logical deduction using the built-in axioms. Section 3.7.2 identifies two additional modes of retrieval that are useful in natural language processing. Viewing a matcher as a set of axioms proves advantageous once again; it leads to the conception of additional retrieval modes as other kinds of logical operations with the same built-in axioms and hence within the same semantic theory.

3.7.1 Time

While knowledge involving time was not expressly excluded from previous representations, no support for reasoning about such knowledge was provided either. We are currently incorporating a model of temporal knowledge based on time intervals (Allen, 1981a; Allen, 1981b). This involves introducing a new set of objects into our domain of discussion, namely time intervals. There is not space to discuss the nature of time intervals here, but see (Allen, 1981a). For our purposes here, we can assume there is simply one relationship that can hold between intervals; one interval can be during another. There are actually 13 mutually exclusive relationships between intervals, all of which require an analysis along the lines shown in this paper. This particular relation was chosen because there are inferences involving it that we would like the matcher to perform automatically. We begin to characterize these inferences

by defining a predicate, DURING(a,b), that is true only if interval a is wholly contained in interval b. The first axiom states that DURING is transitive:

(A.10) $\forall a,b,c\ \text{DURING}(a,b) \wedge \text{DURING}(b,c) \rightarrow \text{DURING}(a,c)$

Given these time intervals, any object, event, or relation can be qualified by a time interval as follows: for any untimed concept x, and any time interval t, there is a timed concept consisting of x viewed during t which is expressed by the term TCONCEPT(x,t). This concept is of type TIMED(Tx), where Tx is the type of x. Thus we require a type hierarchy of timed concepts that mirrors the hierarchy of untimed concepts.

Now, if we want the retriever to automatically infer that if relation R holds during an interval t, then it holds in all subintervals of t, we need the following built-in axiom:

(A.11) $\forall p,t,t',c\ \text{HOLDS}(\text{TCONCEPT}(p,t),c) \wedge \text{DURING}(t',t) \rightarrow \text{HOLDS}(\text{TCONCEPT}(p,t'),c).$

Thus, if the user queries if p holds today, and the knowledge base knows that p holds all week long, the query will succeed.

An axiom similar to (A.11) holds for the IS-REAL predicate operating on timed objects.

(A.12) $\forall c,t1,t2,o\ \text{IS-REAL}(\text{TCONCEPT}(o,t2),c) \wedge \text{DURING}(t1,t2) \rightarrow \text{IS-REAL}(\text{TCONCEPT}(o,t1),c).$

Such an axiom does not in general hold for events. In fact, events need to be divided into two classes dependent on their relationship to the time over which they occur. *Continuous events* consist of some property holding (or being maintained) over a time interval, whereas *atomic events* consist of some change in the world. The most important characteristic of a continuous event is that if it occurs during some time interval, then it occurs during all subintervals. For example, if a jogger runs around a track for an hour then he is running around a track during all periods within that hour. Atomic events, on the other hand, have the opposite characteristic. If a robot moves a block from point A to point B during some time interval, then it has *not* moved the block from A to B during any period within that time interval. Hence, there are two axioms, one for each class of events:

(A.13) $\forall c.t1.t2.e\ TYPE(e,CONTINUOUS\text{-}EVENT) \wedge OCCURS(TCONCEPT(e,t2),c) \wedge DURING(t1,t2) \rightarrow OCCURS(TCONCEPT(e,t1)),c)$

(A.14) $\forall c.t1.t2.e\ TYPE(e,ATOMIC\text{-}EVENT) \wedge OCCURS(TCONCEPT(e,t2)),c) \wedge DURING(t1,t2) \rightarrow \sim OCCURS(TCONCEPT(e,t1)),c).$

Thus we have extended our representation to handle simple timed concepts with only a minimal amount of analysis.

3.7.2 Retrieval Modes

One of the more interesting consequences of our approach is that it has led to identifying various modes of retrieval that are necessary to support a natural language comprehension task. We have considered so far only one mode of retrieval, which we call *provability mode*. In this mode, the query must be shown to logically follow from the built-in axioms and the facts in the knowledge base. While this is the primary mode of interaction, others are also important.

In *consistency mode*, the query is checked to see if it is logically consistent with the facts in the knowledge base with respect to the limited inference mechanism. While consistency in general is undecidable, with respect to the limited inference mechanism it is computationally feasible. Note that, since the retriever is defined by a set of axioms rather than a program, consistency mode is easy to define.

Another important mode is *compatibility mode*, which is very useful for determining the referents of description. A query in compatibility mode succeeds if there is a set of equality and inequality assertions that can be assumed such that the query would succeed in provability mode. For instance, suppose someone refers to an event in which John hit someone wearing a hat. The event might be described as E1 with V1 as the victim.

(34) $TYPE(E1,HIT\text{-}EVENT) \wedge ROLE(E1,AGENT,JOHN) \wedge ROLE(E1,OBJECT,V1) \wedge TYPE(WEAR1,WEAR\text{-}HAT\text{-}RELATION) \wedge ROLE(WEAR1,AGENT,V1) \wedge TYPE(V1,PERSON)$

To be precise we would need to assert that WEAR1 holds at some time in some context, and similarly that E1 occurred. But let us ignore such details.

If we want to find possible referents in memory for E1, and possibly V1 as well, we could try to retrieve all x such that x = E1. But this will not succeed in provability mode since V1 is probably not known elsewhere in the system, and thus cannot be shown equal to any object of a hit event. In consistency mode, (11) will probably succeed vacuously: the facts (10) and (11) would be consistent with an empty knowledge base. What we want to retrieve is all x that *could* be equal to E1 given what we know. Thus if we know that E3 is the event of John hitting Sam, where Sam was wearing a hat, then as far as we know, E1 could be equal to E3. In other words, it is consistent to assume that E1 = E3 and V1 = SAM. If Sam is known not to be wearing a hat, however, then we would not be able to assume that V1 = SAM and hence unable to assume E1 = E2.

We are currently attempting to formalize compatibility mode using Reiter's (1980a) non-monotonic logic for default reasoning. Since retrievals in this mode are inherently expensive, we are investigating methods – similar to those employed by Grosz (1977) – of limiting search to a restricted context.

3.8 Conclusions

We have argued that the appropriate way to design knowledge representations is to identify those inferences that one wishes to facilitate. Once these are identified, one can then design a specialized limited inference mechanism that can operate on a data base of first order facts. In this fashion, one obtains a highly expressive representation language, as well as a well-defined and extendable retriever.

This approach has been demonstrated by outlining a portion of the representation used in ARGOT, the Rochester Dialogue System (Allen, Frisch and Litman, 1982).

We have implemented a version of this system in HORNE (Allen and Frisch, 1981; Frisch, Allen and Giuliano, 1982), a LISP-embedded logic-programming language. Currently the system provides the inheritance hierarchy, simple equality reasoning, contexts, and temporal reasoning with the DURING relation. In conjunction with this representation is a front-end that provides many abbreviations and facilities for system users. For instance, users can specify what context and times they are working with respect to, and then omit this information from their interactions with the system. Also, using the abbreviation

conventions, the user can describe a relation and events without explicitly asserting the TYPE and ROLE assertions.

References

Allen, J.F., "An interval-based representation of temporal knowledge," *Proc.*, 7th IJCAI, Vancouver, BC, 1981a.

Allen, J.F., "What's necessary to hide?: Reasoning about action verbs," *Proc.*, 19th Annual Meeting, Assoc. for Computational Linguistics, 77-81, Stanford U., 1981b.

Allen, J.F. and A.M. Frisch, "HORNE user's manual," Internal Report. Computer Science Dept., U. Rochester, 1981.

Allen, J.F. and A.M. Frisch, "What's in a semantic network?" *Proc.*, 20th Annual Meeting, Assoc. for Computational Linguistics, U. Toronto, 1982.

Allen, J.F., A.M. Frisch and D.J. Litman, "ARGOT: The Rochester dialogue system," *Proc.*, 2nd National Conference on Artificial Intelligence, Carnegie-Mellon U., 1982.

Bobrow, D.G. and T. Winograd, "An overview of KRL, a knowledge representation language," *Cognitive Science 1*, 3-46, 1977.

Bowen, K.A. and R.A. Kowalski, "Amalgamating language and metalanguage in logic programming," in K. Clark and S.A. Tarnlund (Eds). *Logic Programming*. New York: Academic Press, 1982.

Brachman, R.J., "On the epistemological status of semantic networks," in N.V. Findler (Ed). *Associative Networks*. New York: Academic Press, 1979.

Brown, F.M., "Towards the automation of set theory and its logic," *Artificial Intelligence 10*, 281-316, 1978.

Charniak, E., "A common representation for problem-solving and language-comprehension information," *Artificial Intelligence 16*, 225-255, 1981a.

Charniak, E., "The case-slot identity theory," *Cognitive Science 5*, 1981b.

Clark, K.L., "Negation as failure," in H. Gallaire and J. Minker (Eds). *Logic and Data Bases*. New York: Plenum Press, 1978.

References

Clark, K.L. and F.G. McCabe, "The control facilities of IC-PROLOG," in D. Michie (Ed). *Expert Systems in the Micro Electronic Age*. Ediburgh: Edinburgh U. Press, 1979.

Cohen, P.R., C.R. Perrault and J.F.Allen, "Beyond Question-Answering," in W.G. Lehnert and M.H. Ringle (Eds). *Strategies for Natural Language Processing*. Hilsdale, N.J.: Lawrence Erlbaum, 1982.

Davidson, D., "The logical form of action sentences," in N. Rescher (Ed). *The Logic of Decision and Action*. Pittsburgh, PA: U. Pittsburgh Press, 1967.

Fikes, R.E. and N.J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, 2, 189-205, 1971.

Fillmore, C.J., "The case for case," in E. Bach and R. Harmes (Eds). *Universals in Linguistic Theory*. New York: Holt, Rinehart and Winston, 1968.

Findler, N.V. (Ed). *Associative Networks: Representation and Use of Knowledge by Computers.* New York: Academic Press, 1979.

Frisch, A.M., "A formal study of knowledge representation and retrieval," Ph.D. Thesis Proposal, Computer Science Dept., U. Rochester, 1981.

Frisch, A.M. and J.F. Allen, "Knowledge retrieval as limited inference," in D.W. Loveland (Ed). *Lecture Notes in Computer Science: 6th Conference on Automated Deduction*. New York: Springer-Verlag, 1982.

Frisch, A.M., J.F. Allen and M. Giuliano, "An Overview of the HORNE logic programming system," Submitted to *Logic Programming Newsletter.* December 1982.

Green, G., Theorem-proving by resolution as a basis for question-answering systems." in B. Meltzer and D. Michie (Eds). *Machine Intelligence 4.* Ediburgh: Edinburgh University Press, 1969.

Grosz, B.J., "The representation and use of focus in dialogue understanding," Technical Note 151, SRI International, July 1977.

References

Haas, A.R., "Planning mental actions," Ph.D. thesis, TR 106, Computer Science Dept., U. Rochester, 1982.

Hayes, P.J., "The logic of frames," in D. Metzing (Ed). *Frame Conceptions and Text Understanding*. Walter de Gruyter & Co., 1979.

Hendrix, G.G., "Encoding knowledge in partitioned networks," in N.V. Findler (Ed). *Associative Networks*. New York: Academic Press, 1979.

Kowalski, R.A. *Logic for Problem Solving*. New York: North Holland, 1979.

Levesque, H. and J. Mylopolous, "A procedural semantics for semantic networks," in N.V. Findler (Ed). *Associative Networks*. New York: Academic Press, 1979.

Loveland, D.W. *Automated Theorem Proving: A Logical Basis*. Amsterdam: North-Holland, 1978.

Luckham, D.C. and N.J. Nilsson, Extracting information from resolution proof trees. *Artificial Intelligence 2*, 27-54, 1971.

Meltzer, B., "Theorem-proving for computers: Some results on resolution and renaming," *Computer Journal 8*. 341-343, 1966.

Moore, R.C., "Reasoning about knowledge and action," Ph.D. thesis, MIT, 1979.

Nilsson, N.J. *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga Publishing Co., 1980.

Norman, D.A. and D.G. Bobrow, "On data-limited and resource-limited processes," *Cognitive Psychology 7*. 44-64, 1975.

Pereira, L.M., and A. Porto, "Selective backtracking for logic programs," *5th Conference on Automated Deduction Proceedings*, Springer-Verlag, 1980.

Perlis D., "Language, computation, and reality," Ph.D. thesis and TR95. Computer Science Dept., U. Rochester, 1981.

Reiter, R., "A logic for default reasoning," *Artificial Intelligence 13*, 81-132. 1980a.

Reiter, R., "Equality and domain closure in first-order databases," *Journal of the ACM 27*, 235-249, 1980b.

Robinson, J.A. *Logic: Form and Function.* New York: North Holland, 1979.

Robinson, J.A. and E.E. Sibert, "LOGLISP: an alternative to PROLOG," in J.E. Hayes, D. Michie and Y-H Pau (Eds). *Machine Intelligence 10.* Chichester, England: Ellis Horwood, 1982.

Shapiro, S.C., "A net structure for information storage, deduction and retrieval," *Proc.*, 2nd IJCAI, 1971.

Shapiro, S.C., "The SNePS semantic network processing system," In N.V. Findler (Ed). *Associative Networks.* New York: Academic Press, 1979.

Stefik, M.J., "Planning with constraints," *Artificial Intelligence 16*, 111-140, 1981.

Tarjan, R.E., "Efficiency of a good but not linear set union algorithm," *JACM 22*, 215-225, 1975.

Van Emden, M.H., "Programming with resolution logic," in E.W. Elcock and D. Michie (Eds). *Machine Intelligence 8.* Chichester, England: Ellis Horwood, 1977.

Warren, D.H.D. and F.C.N. Pereira, "An efficient easily adaptable system for interpreting natural language queries," DAI Research Paper 155, Dept. Artificial Intelligence, U. Ediburgh, 1981.

Weyhrauch, R.W., "Prolegomena to a theory of mechanized formal reasoning." *Artificial Intelligence 13*, 133-170, 1980.

Woods, W.A., "What's in a link: Foundations for semantic networks," in D.G. Bobrow and A.M. Collins (Eds). *Representation and Understanding.* New York: Academic Press, 1975.

END

DATE
FILMED

9 83

DTIC